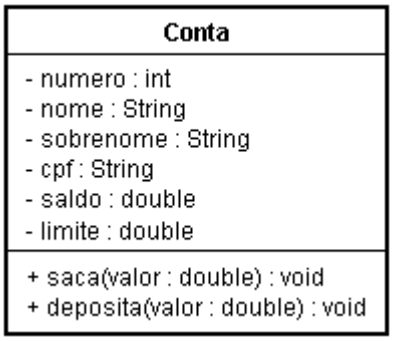
CAPÍTULO 8
CONSTRUTOR

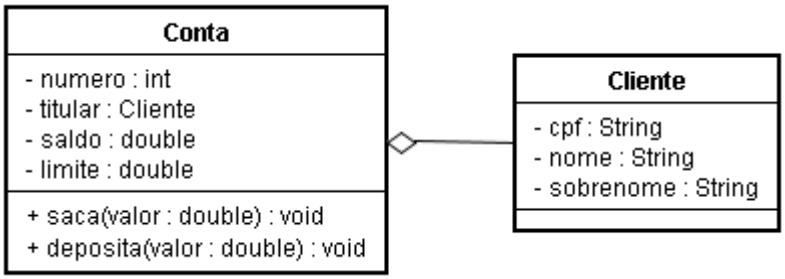
Permite que um determinado trecho de código seja executado toda vez que um objeto é criado, ou seja, quando precisamos inicializar propriedades de um objeto de uma classe no momento em que este é criado, usamos um construtor.

Quando usamos a palavra chave NEW, estamos construindo um objeto e, sempre quando o NEW é chamado, ele executa o construtor da classe. Até agora, as nossas classes não possuíam nenhum construtor. Quando não declaramos construtor na classe, o Java cria um construtor default, que não recebe nenhum argumento e o corpo dele é vazio. A partir do momento que você declara um construtor, o construtor default não é mais fornecido.

Imagine que aumentamos a nossa classe Conta e adicionamos nome, sobrenome e cpf do cliente, dono da conta.



Começaríamos a ter muitos atributos e, se pensarmos direito, uma CONTA não tem NOME, nem SOBRENOME e CPF, quem tem esses atributos é um CLIENTE. Então podemos criar uma nova classe e fazer uma COMPOSIÇÃO:



E dentro do MAIN da classe PRINCIPAL:

```
public class Principal {

    public static void main(String[] args) {
        Cliente titular = new Cliente();
        titular.setNome("Horácio");
        Conta minhaConta = new Conta();
        minhaConta.titular=titular;
        //...
    }
}
```

Aqui, simplesmente houve uma atribuição. O valor da variável `TITULAR` é copiado para o atributo titular do objeto ao qual `MINHACONTA` se refere. Em outras palavras, `MINHACONTA` tem uma referência ao mesmo `CLIENTE` que `TITULAR` se refere, e pode ser acessado através de `MINHACONTA.TITULAR`.

Mas, e se dentro do código não déssemos `NEW` em Cliente e tentasse acessá-lo diretamente?

```
public class Principal {
    public static void main(String[] args) {
        Conta minhaConta = new Conta();
        minhaConta.titular.setNome("Horácio");
        System.out.println(minhaConta.titular.getNome());
        //...
    }
}
```

Quando damos `NEW` em um objeto, ele o inicializa com seus valores default: 0 para números, false para boolean e null para referências. Null é uma palavra chave em Java, que indica uma referência para nenhum objeto.

Se, em algum caso, você tentar acessar um atributo ou método de alguém que está se referenciando para `NULL`, você receberá um erro durante a execução:

```
Exception in thread "main" java.lang.NullPointerException
        at Principal.main(Principal.java:10)
```

Da para perceber, então, que o `NEW` não traz um efeito cascata, a menos que você dê um valor default ou use construtores.

O construtor de uma classe é um bloco declarado com o mesmo nome da classe:

```
public class Conta {

    //...

    //Construtor
    public Conta() {
        System.out.println("Construindo uma conta.");
    }

    //...
}
```

Então, quando fizermos:

```
Conta minhaConta = new Conta();
```

A mensagem "Construindo uma conta." aparecerá. É como uma rotina de inicialização que é chamada sempre que um novo objeto é criado.

Poderíamos instanciar o cliente dentro do construtor da classe CONTA para evitarmos o NULLPOINTEREXCEPTION:

```
public class Conta {

    Cliente titular;
    //...

    //Construtor
    public Conta() {
        titular = new Cliente();
    }

    //...
}
```

O interessante é que um construtor pode receber um argumento, podendo assim inicializar algum tipo de informação:

```
public class Conta {

    Cliente titular;
    //...

    //Construtor
    public Conta(Cliente titular) {
        this.titular=titular;
    }

    //...
}
```

Esse construtor recebe o titular da conta. Assim, quando criarmos uma conta, ela já terá um determinado titular.

```
Cliente titular = new Cliente();
titular.setNome("Horácio");

Conta minhaConta = new Conta(titular);
System.out.println(minhaConta.titular.getNome());
```

8.1. MOTIVAÇÃO PARA USAR CONSTRUTORES

Se tudo estava funcionando até agora, para que utilizamos um construtor? A ideia é bem simples. Se toda conta precisa de um titular, como obrigar todos os objetos que forem criados a ter um valor desse tipo? Basta criar um único construtor que recebe essa String!

O construtor se resume a isso! Dar possibilidades ou obrigar o usuário de uma classe a passar argumentos para o objeto durante o processo de criação do mesmo. Por exemplo, não podemos abrir um arquivo para leitura sem dizer qual é o nome do arquivo que desejamos ler. Portanto, nada mais natural que passar uma String representando o nome de um arquivo na hora de criar um objeto do tipo de leitura de arquivo, e que isso seja obrigatório.

Outro motivo para criação de construtores: facilidade. Às vezes, criamos um construtor que recebe diversos argumentos para não obrigar o usuário de uma classe a chamar diversos métodos do tipo "set". Podemos forçar que uma classe Cliente receba no mínimo o CPF, dessa maneira um Cliente já será construído com um CPF válido.

Resumindo, criar um Construtor é uma forma de garantir que, quando o operador `NEW` for utilizado, o objeto terá à disposição as funcionalidades mínimas necessárias, garantindo também que nenhum atributo do objeto terá um valor arbitrário, que ocasione erros difíceis de detectar.

8.2. JAVABEANS

Javabeans são classes que possuem o construtor sem argumentos e métodos de acesso do tipo get e set como, por exemplo, as classes de modelo de nossos projetos. A especificação JavaBeans é muito grande e mais informações sobre essa vasta área que é a base dos componentes escritos em Java pode ser encontrada em:

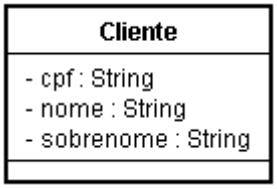
http://java.sun.com/products/javabeans

8.3. EXERCÍCIOS: Construtor

1) Importe o projeto no Eclipse:
   a. Acesse o diretório `PROJETOS` no moodle;
   b. Copie a pasta **ds1-controle-bancario** para sua área de trabalho;
   c. Abra o Eclipse;
   d. Acesse o menu **File ➔ Import ➔ General Existint Projects into Workspace**

e. Avance, clique no botão **Browse** e selecione a pasta do projeto **ds1-controle-bancario**
f. Clique em **Finish**

2) Crie a classe Cliente com seus atributos privados e seus *getters* e *setters*:



3) Abra a classe Conta e acrescente o atributo titular do tipo Cliente:

```
public class Conta {

    private double saldo;
    private double limite;
    public Cliente titular;

    //getters e setters omitidos
    //métodos específicos omitidos
}
```

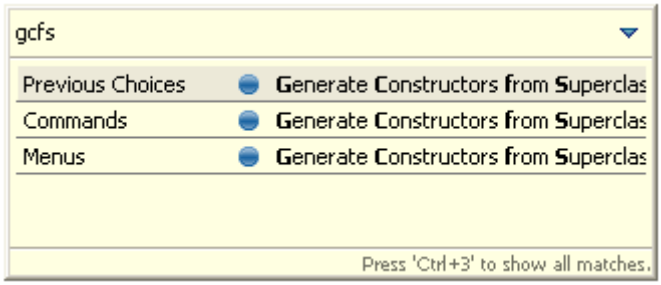
4) Crie a classe Principal com o método main():

```
public class Principal {
    public static void main(String[] args) {
        Conta minhaConta = new Conta();
        minhaConta.titular.setNome("Horácio");
        System.out.println(minhaConta.titular.getNome());
    }
}
```

5) Execute a classe Principal e veja o resultado:

```
Exception in thread "main" java.lang.NullPointerException
        at Principal.main(Principal.java:10)
```

6) Abra novamente a classe Conta e acrescente o construtor:
a. Pressione **CTRL + 3**
b. Digite *gcfs* que é abreviação para GENERATE CONSTRUCTOR FROM SUPERCLASS e pressione **ENTER**



c. Desative a opção **Omit call to default constructor super()**

d. Pressione **OK**

7) Dentro do construtor, instancie o objeto titular:

```
public Conta() {
    titular = new Cliente();
}
```

8) Salve a classe Conta e execute novamente a classe Principal.

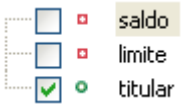
9) Apague o construtor criado.

~~public Conta() {~~
~~titular = new Cliente();~~
~~}~~

10) Altere o modificador public do atributo titular para private e gere o get e set para ele.

~~public~~ private Cliente titular;

11) Criaremos agora um construtor que recebe um cliente:
a. Pressione CTRL + 3
b. Digite *gcuf* que é abreviação para **Generate Constructor Using Fields** e pressione **ENTER**
c. Deixe marcado apenas a opção titular:



d. Desative a opção **Omit call to default constructor super().**
e. Pressione **OK**.

12) Altere a classe Principal para que, ao criarmos uma conta, ela já terá um determinado titular.

```
public class Principal {
    public static void main(String[] args) {
        Cliente titular = new Cliente();
        titular.setNome("Horácio");
        Conta minhaConta = new Conta(titular);
        System.out.println(minhaConta.getTitular().getNome());
    }
}
```

Referência Bibliográfica:

Apostila Caelum - http://www.caelum.com.br/
http://www.globalcode.com.br/home
http://www.k19.com.br/downloads/apostilas-java
http://www.algaworks.com/
Barnes, David J. e Kolling, Michael. Programação Orientada a Objetos com Java. Pearson
Arnold, Ken, Gosling, James e Holmes, David. A Linguagem de Programação Java. Bookman
Harvey M. Deitel, Paul J. Deitel. Java Como Programar. 6ed. Pearson
Flanagan, David. Guia Essencial do Java. Bookman
Martin, Fowler. UML Essencial.
Livro de Informática – Centro Paula Souza